[illegible]

Wednesday, October 6, 2010 at 7:54PM

There is a very significant difference between a game mechanic (or collection thereof) and a finished game. It's not the quality of the game mechanic, or the way it plays, or the polish you apply to it. It's all the other stuff you just don't care about when you're prototyping or concentrating on The Next Great Idea.

As dull as it may feel in comparison, finishing a game makes a world of difference to the "game" play itself. Take, use and implement just one of the items on this list and then tell me your game isn't better off because of it. Seriously, the first time time I added a pause feature to a prototype I actually thought to myself, "Wow, it's like I made a real game!"

So - checklist, design input, verification or otherwise - if you want to make sure your game is finished, you need this list. I'm assuming the production of the Secret Sauce is under control.

Mise

Prepare each of the ingredients below, as soon as you possibly can. This recipe's all mise.

Ingredients

1. Pause

Simple - a button that when pressed pauses the game, wth a corresponding "resume" button. Ideally:

- It should pause immediately.
- The resume should resume gameplay immediately (expect for situations where it's awkward for the player to reconfigure their posture between the press of the resume button and normal gameplay - e.g. a console racer that uses the start button for pause/resume and a face button for gas).
- It should be located as per the norms of the platform (iPhone - in a corner; Xbox 360 - start button).
- Resume should be triggered by the same interaction (press start to resume, tap same corner etc).

about

blog

login

rss

apps

Ludum Racer (challenge blog)

Dead West

Find the Gumbolts

Bucket O' Mullets

Pheasant Plucker

archive

admob (1) ai (1) appstore (1) art (3) assets (2) chaos (1) code (2) commentary (1) creativecommons (4) Dead West (2) design (3) fantasy (1) free (2) game mechanics (1) games (2) gumbolts (6) how to (2) ideas (2) idevblogaday (2) inspiration (1) iOS 4.0 (2) japan (1) katana (1) links (3) modernsoldier (1) ninja (1) pax (1) pheasant plucker (3) pictures (1) productivity (1) rendering (1) resource (1) sales (1) shinobi (1) shuriken (1) sprite (3) texture tiling (1) UIKit (1) video (1) visualization (1) work in progress (2) xcode (1)

from our art dept...

- Audio and sound effects should pause immediately (no ambient noise or background music in pause screens please).
- Unless it provides an unwanted advantage, let the player see the frozen game screen in the background of the pause screen

2. Menus

Do this early. Avoid hacks and workarounds for accessing normally menu-driven features. It's easy to sketch out a layout of your menus and implement the navigation between them early on. Ideally:

- Menus should be immediately responsive to user interaction and load quickly (see almost any DVD or Blu-ray for an example of how not to do this).
- Menus should be easy and intuitive to use (that obscure setting might be quirky the first time, but it just gets irritating).

3. Save and load

Yes, this is hard. But liklihood is that you're going to need it. Get it in there early. Ideally:

- Saves should happen when you expect them to (Bungie's insistence on making the Halo player actually select "Save" despite having passed "checkpoints" since the last save is illogical and gets me every time -- it's still doing it in Reach).
- Saving should be quick - please don't make save game "names" mandatory on a console device, for example.
- It should be easy for the player to understand what the consequences of loading and saving are (i.e. do I lose current progress, can I resume from an earlier save etc). If Quantic Dream can do it for Heavy Rain, you can.
- There should be support for multiple save profiles. Some platforms provide for this briliantly (360). Some do not (I'm looking at you Steve). Worst case, Nintendo's model for most Mario games works pretty well.

4. Reset / restart

Unless you're specifically wanting to prevent it as part of the game's design, make it easy for the player to reset/restart since the last significant juncture (level, checkpoint, whatever). Ideally:

- Resets should be independent of the save/load feature - some players will want to reset a level regardless of where they loaded it from.

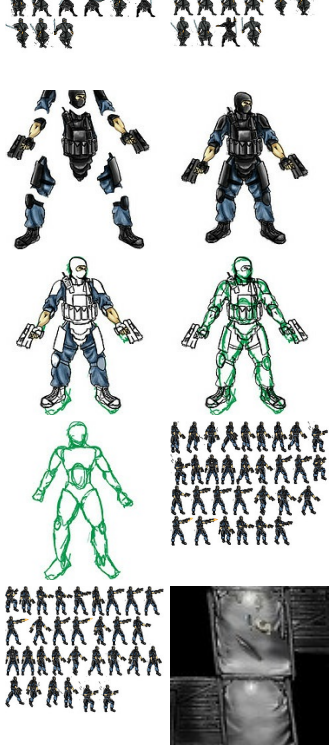
5. Platform-expected norms -- i.e. iOS resume / multitasking

This does apply to other platforms - but the iOS 4.0-introduced multi tasking has reinforced players' expectations around app switching and resume. If you've done (3.) this shouldn't be a problem.

6. Sound effects

I deliberated including music and art on this list -- but to be honest it's difficult to argue that they are really required to make a game a "game" as such. SFX are a different matter however: they provide one of the two primary/common sensory feedback mechanisms (visual being the other). Ideally:

- SFX should be optional - in most cases it should be possible to play the game without any SFX.



- SFX should enhance the experience, not distract from it.
- SFX should provide feedback to the player -- on a touch-screen device, for example, they help you compensate for the lack of tactile feedback on button/screen presses.

7. A title

This is all I'll say about marketing here -- and I'm not really including it for that reason, it just it normally gets lumped in "marketing". Your game needs a title. It doesn't matter what you call it -- you can change it later. You'll be amazed by how much easier it is to talk about your game when it has a name.

8. An indication of progress

The player has to understand what they have accomplished by playing the game. This is better explained by a list of what progress might look like:

- High sores
- Accessing a new level
- Unlocking an item
- Being rewarded with an achievement

9. Instructions and help

Think about a player picking this up for the first time - what do they need to know? Ideally:

- Controls should leverage the expectations of platform (left stick move; right stick look etc).
- Objectives should be clear when they need to be (it's okay if you want the player to figure this out -- but that has to be a core part of your game's design)
- There should be an easy fallback for players -- if they don't know what do where is the first place they're going to start looking? That's where you should put the help and instructions.
- Dynamic help is useful but it can get tiresome -- detect abnormal use (like the player hasn't jumped yet) and gracefully display non-intrusive instructions ("press A to jump")
- If you provide a structured training environment (whether this is a couple of dialogs or a training mode) make sure you clearly indicate length and relative progress.

10. The Secret Sauce

You did this already, right?

Gareth Jenkins | Post a Comment | Share Article | Permalink | tagged design, how to, idevblogaday

Wednesday, September 29, 2010 at 5:54PM

As the first post in our #idevblogaday Wednesday spot, I thought I'd take the opportunity to expand on a few points relating to the management and getting things done of software development with an entrepreneurial slant.

This is pitched at those involved in the practice of software design and development - primarily for those working alone or managing small teams. However, it's easily applied to people doing similar in other fields - particularly complex creative projects. It jumps about a bit between personal productivity and project design and decisions.

The inspiration for this list comes directly from a couple of videos in which Jerry Kaplan discusses the 5 biggest mistakes entrepreneurs make and the 5 critical skills entrepreneurs need. His points are related to business development and the management of people specifically, but the core of them can be applied easily to the creation of things, especially in the realm of software development.

I manage and lead a small team that makes computer games (iOS computer games at present) -- and the points I make below are all drawn directly form my experience and development doing that, and from having managed a variety of teams of developers and other creatives in the production of a variety of software projects, business applications and services. It all works the same, it's just that some of it (the games development bit particularly) is more fun.

A lot of the points below can be applied to different aspects and levels of software development and design, but I've mostly avoided the people-management applications as there's no point in repeating Kaplan's original (and very useful) observations.

1. Set a clear mission, goals and objectives at every level

Establish how you intend to measure success - at every level. How do I know if this feature is complete? How do I test that this bug is resolved? What should this button do? What problem is this application trying to solve? In what way is this game fun for the player?

Without a clear objective and measure of success, not only is it impossible to efficiently move from one task to the next, it's difficult to guarantee that the work you're putting in is even required in the first place.

2. Don't try to be clever

Being clever for the sake of it does nothing but make you look like an ass. And it almost certainly confuses the requirements or expectations of anything you create.

When you come back next week to write a parser for that set of data structures you just designed, what would you prefer - an obfuscated mess of techniques you don't fully understand, or a solid, transparent solution to the problem they were intended to solve?

Users of your software don't care how smart you are. In fact, they would really prefer to feel that they are the smart ones for choosing you in the first place. If you want to show off, go join the circus. If you want to impress users, deliver on your promises (see #1 Objectives).

3. Don't do it for the money

Do it because you want to, and worry about the rest later. If you genuinely want to make software, then make your decisions in that context.

If you can figure out how to find and utilise resources (your time, skills and passion count) and convert them into value, what you do inherently has value in and of itself. If you can't figure out how to make money at that point, go ask someone else - others are more than happy to help if properly recognised or rewarded for their input.

4. Share your wealth and skills

Kaplan makes the observation that "equity is like shit: if you pile it up, it just smells bad; if you spread it around lots of wonderful things grow." This can be directly applied to your most precious resource -- your time.

If you concentrate on any one piece of the software development process or on a specific aspect of one stage of that process you will almost certainly fail to meet your objectives. Just because you love to perfect the game mechanic doesn't mean you shoudn't be paying equal attention to the UI, or the audio, or the help system, or the game's marketing material... and on, and on.

5. Use what you need, not what you like

Maybe I've spent too long in the enterprise software world, but this seems to be apparent in almost every software development or delivery project I've been involved in.

It's simple: use the tools that get things done.

Do not use something (a platform, a method, a person, an IDE, whatever) because you like it -- remember this has got nothing to do with you -- keep your objectives in mind (#1 Objectives) and choose what gets the job done.

6. Know when to let go

Just because you thought it was a good idea a week ago, doesn't mean the same holds true right now. Your agility is one of your key strengths - adapt, change and -- if you have to -- let go of what you don't need.

7. Telescope as required

Every single one of us has spent an afternoon so absorbed in detail that by the end of it you're unsure of how you got there and, more importantly, why you were there in the first place.

If you're not consciously doing it already, identify and refine your ability to telescope: You need to be able to pick up on the detail of a problem, apply your skills and solve it with a constant view on its relationship to the wider context.

8. Lead and make decisions when you need to

You have to make decisions in order to move forward and if you want to be in any way efficient about doing so, you need to make them at the right time - not too early and not too late.

Don't try and anticipate problems for the sake of doing so and don't avoid making a decision because it changes something that came before (see #6 Letting go).

As Kaplan points out, leadership is often defined as the ability to build consensus in the face of uncertainty. Software design and development presents infinitely uncertain paths - you need to have the confidence that you can consistently pick them as you need to.

Update: For those of you who saw the original title -- I was planning on 10, but settled on 8. I guess 9 and 10 could be "pay attention" and "be concise."

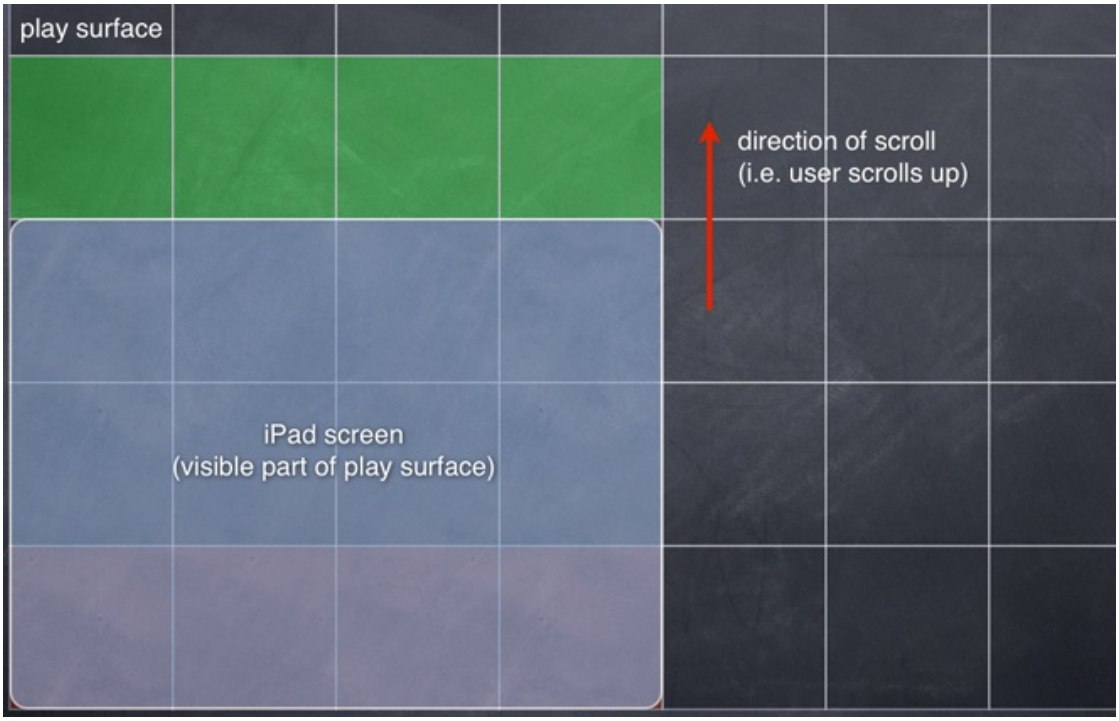
Tuesday, September 28, 2010 at 3:45PM

Dead West, an iPad game which we recently introduced here uses a forced-perspective top-down view of the game world - with, what we originally thought trivial - painted background.

Because the goal of the technical design was to provide for a high quality painted background that will spread beyond the maximum texture capacity supported by the iPad (2048x2048), we needed to design a rendering engine that will be able to take multiple tiles and draw them as one seamless, scrollable background on the iPad screen. After a holy-crap-this-is-harder-than-we-thought stage, we ended up with a design that meets most of our needs, with additional possibility for optimization if needed.

The process we are currently using (Texture Tiling) shouldn't be confused with Tile Mapping. Tile maps are normally created from a texture atlas where you can draw many objects by reusing elements on the same atlas. This allows you to draw many objects on the screen using one batch (i.e. no texture swapping).

The Texture Tiling engine's job is to take tiles from the disk and make them available in VRam. Once in VRam, we use those textures to build a bigger texture using an Frame Buffer Object (FBO). An FBO is a render-to-texture technique often used for special effects. In our context, we use it to cache previously loaded textures, so they can be unloaded when not required. The first time we create the FBO it's a bit time consuming because we need to get all of the tiles (currently 256x256px) from disk and render them one after the other into the FBO. But further updates are much faster since we can reuse the FBO texture itself and only add to it the new textures that are coming into view.



In the above outline, the green tiles will be rendered as the visible part of the play surface moves up, and the red ones ultimately unloaded once they move out of the visible view.

This techniques is working well when we don't have too many textures to load from disk after an update. The game plays in landscape orientation, so, for example, we can render and scroll an 8000x768px play surface with relative ease using this technique. Every time we need to load a new zone (textures that are not yet loaded from disk), we load a maximum of 3 textures (3x256==768px==iPad screen height in landscape orientation). Similarly, a 1024x8000px play surface results in a maximum load of 4 textures.

A square play surface of 2048x2048px is the natural size of a fully loaded FBO and doesn't incur any cost at all. This is by far the best bang for the buck size vs performance.

Maps over 2048x2048px cause some performance headaches for the engine. The algorithm we're currently using loads a full FBO width and/or height on zone change. This results in a load of between 8 and 16 textures. Whilst we're not doing so currently, we'll be optimizing this to only update the visible play surface area, reducing load count of textures to between 4 and 8.

Another possible optimization would be to cache loaded sections instead of querying the map for loaded sections based on the current area (we're currently assuming this isn't a bottleneck though).

Beyond that, we'll assess performance and if necessary look to optimize further by asynchronously loading textures from disk when getting close to an unloaded zone. If we spit the FBO in 4 quadrants, such that if we're viewing the top left quadrant we should start loading textures that will be on the right and bottom edges. When moving the player(s) around, this should result in an almost unnoticeable load sequence. However, we do not expect to see a noticable improvement when moving around by scrolling (which we also support) as it's likely the user will scroll through sections too quickly. If required, we could compensate for this by not allowing scrolling outside of loaded/navigated zones or by adding fog-of-war-like effects for zones that are still loading.

We'll report back with some performance data as and when we can -- as well as a more technical description of the engine itself once it's matured a little.

The design and engine above was all done by our rendering/gfx/misc coder, Marc Hebert -- who can be also be found at his own blog http://shadhex.blogspot.com or more frequently on Twitter at http://twitter.com/shadhex.

Gareth Jenkins | Post a Comment | Share Article | Permalink | tagged Dead West, code, design, rendering, texture tiling

[illegible]

Friday, September 24, 2010 at 1:36PM

Earlier this year we were briefly involved in the design and early production of a Chaos-like game, inspired directly by Julian Gollop's popular original Chaos: The Battle of Wizards.

If you're not familiar with the original check out the many resources linked from the above wikipedia article and the original game rules.

Before the project was cancelled, we produced a version 1 AI design - which, unfortunately, never got used (or reviewed for that matter). So, in all it's unedited,

incomplete glory, here is a design document for a Chaos AI. Apologies for the variable formatting - it's come via Google Docs > HTML > embedded CSS to this post.

Use it for what you like - and if you do fancy reviewing, editing or improving on it, do so and let us know in the comments and we'll keep it up to date.

AI Design for a Chaos-like game

Notes

This assumes game rules are enshrined in the interface functions provided to the AI player/code, but is written in such a way that it should behave in legally.

This is (should be) modular such that individual methods and calculations can be adjusted independently.

It assumes a set of constants that can be adjusted (and managed separately) for each AI player to produce a set of differently-behaving AIs. See AI Type and the **Constants** section for more information.

The design is set up to take positive actions based on the queries it makes to game state. As such, it can be incomplete (insofar as it may not recognise creature, spell or player characteristics or behaviors) but still functional (i.e. will still play turns).

In order to remain performant relatively easily manipulated through data (constant) changes, it performs single turn analysis in two broad stages (tactical and local/per creature). Long running actions (such as multi-turn moves to a target) are managed by creating predictable (within the given set of constants) behaviours that are also affected by previous actions (in the form of last-turn decision memory) that is affected by a weighting constant (see RankTargets and RankTargetsLocally). This allows for one AI player to be very responsive to new circumstances (a nearby threat mid long-running move) whilst another AI player may show determinism to follow through on long-running actions.

Discussion points / things to complete:

1. Should all creatures know ranged attack range of all other creatures? i.e. for inbound proximity (Ranged Defense Proximity, Melee Defense Proximity). A good human player might - but it's not realistic to expect that they always would.

2. How to handle infinite values (i.e. a creature that's not reachable - i.e. in water), significant for the purposes of calculation and multiplication. Options are:

    2.1. Use 0 - will nullify any multiplying factor in relevant calculations
    2.2. Use fixed representation of infinity (e.g. 9999) - will balloon factors in multiplying calculations

The real impact of the above is if a value that could be infinity (any of the proximities potentially) is used in a multiplication (which they are in RankTargetsLocally()), states that return the infinite value will result in either a 0 value or a higher value than anything else (other than other infinities).

One or the other will likely be desirable but a consistent use of potentially infinite

values is preferable. If necessary, the logic could always be switched with (e.g. value==0 in the case of using infinity==0).

EDIT - this is largely irrelevant now - 2.1 isn't a valid option as it's possible to be adjacent to a target and have a proximity query return 0.

3. Handling of non-creature spells not factored into tactical or local decision making (i.e. avoidance of sticky blobs, movement toward citadels etc etc). Casting of these spells also isn't considered (but its not being considered is covered in the current non-specification of SelectAndCastSpell()).

4. How would/should a creature move away from a threat? This is potentially covered in MoveToTarget, option 2 in circumstances that include the creature ranking a very threatening creature as the current target and then acting "scared". This does not cover movement away from a creature/threat that is not the targeted creature, however (i.e. a creature may melee attack creature 1, ranged attack creature 2 but still want to move away from creature 3 or another threat (blob, fire etc).

5. Ranged proximity calculations are not accurate (and could be unpredictably very inaccurate) as they rely on a movement pathfind - obvious alternatives would be computationally expensive.

6. MoveToTarget() needs expansion - both in a tactical sense (making smarter movement decisions) and a practical one (how to accomplish path finding and boundary decisions). See description for more information.

7. How do wizards (and to some extent creatures) stay put / run away (different rank local for wizards?). Affected also by special spells and feature (towers, citadel etc). Separate decision for keeping wizards safe, then weighted against immediate threats (i.e. if it can't move out of range then eval RankTargetsLocally)?

8. How/should creatures make decisions about the threat of a wizard's potential spell casts (i.e. because they can't be evalutated as a simple threat calculation).

Legend

Stub methods / procedures
- written in style MethodName(Argument 1, Argument 2)
- only information relevant at point of mention is included in arguments list

Concepts

DATA

CONSTANTS

META - stuff not done, reminders and suchlike.

1. RankPOPs(): Rank Points of Presence (POPs) - Player's Wizard and Creatures

2. IdentifyTargets(): Identify Targets (up to MAX_TARGETS)

3. RankTargets(): Rank Targets

4. SelectAndCastSpell(): Select & cast spell appropriate spell, insert created creature (if any) into RankedPOPs (no position mechanic as yet - suspect first or last will work best, re-rank is feasible)

5. (optional) IdentifyTargets() as a new creature may affect proximity-based calculations (see IdentifyTargets for options)

6. (if spell cast sucessful) PurgeTargets(): Remove destroyed creatures / wizards

7. (if spell cast sucessful) RankTargets(): Update rankings based on presence of new creature

8. Step creatures (in order of RankedPOPs, but with Wizard moved to last place), and:
    1. TestEngagement(): check if creature is being engaged
    2. RankTargetsLocally(): sort RankedTargets for applicability to this creature
    3. Set CurrentTarget to position 1 on RankedLocalTargets
    4. MoveToTarget()
    5. MeleeAttack()
    6. If CurrentTarget no longer exists, select a new CurrentTarget (position #2 on RankedLocalTargets)
    7. RangedAttack()
    8. If CurrentTarget has not been destroyed, set this creature's LastTarget to CurrentTarget
    9. (if any attack was successful) PurgeTargets()
    10. (if any attack was successful) RankTargets()

RankPOPs

Create sorted list of wizard and creatures (RankedPOPs) for the AI player, with Wizard always first and each creature listed in descending order of Relative Value.

Also needs to factor likelihood of engagement, so rank value should be:
Relative Value - (Number of Adjacent Creatures * ENGAGEMENT_PARANOIA)

IdentifyTargets

Scan entire visible map, then for each found wizard or creature calculate:

Perceived Value * SUM(Simple Distance to POPx from creature * Relative Value of POPx)

Note: the SUM part is evaluated for each of the AI Player's POPs.

Sort this list descending and store top MAX_TARGETS in Targets

Note - actual ranking is done in the RankTargets stage - the approximate calculation of importance shown above exists solely to create a sorted list (such that the top x creatures can be properly considered in RankTargets).

Potential simplification: if it is possible to consider all targets in RankTargets (i.e. for a performance perspective), this step can likely be removed (a simple map scan would suffice).

ALTERNATIVE - work out from POPs (in descending order of importance), going 1 extra hex in radius for each iteration of the POPs list until reaching MAX_TARGETS. Note:
- would need to ignore duplicates
- doesn't necessarily hit all wizards
- no assessment of value / threat / risk imposed by creatures discovered, only considers proximity to important targets

RankTargets

Prepares a ranked list of targets (in order of tactical importance), which can then be sorted locally for each controlled creature, and used for spell casting and wizard movement decisions.

See RankTargetsLocally for usage on a creature level.

Obtain rank for each creature(x) by:
RANK =
(1 / Position in targets list) * TARGET_NEARBY_CREATURES
+ Threat(x, wizard) * WIZARD_PROTECTION
+ SUM(Threat(x, each creature)) * COMPANION_PROTECTION
+ Previous rank (rank score, not position) on RankedTargets list * PARTY_DETERMINATION
+ (1 / Turn Age of x) * ATTENTION_TO_NEW_CREATURES
+ Attention from Opponent * FOCUS_ON_AGRESSORS

Rank should also incorporate target affinity (i.e. targeting creatures belonging to the same wizard), this could be done as a simple distribution, e.g.:

+ (# Affiliated Targets / # Targets) * FOCUS_ON_SINGLE_OPPONENTS *(not yet defined)*

However, it would be more useful to balance it as attention on powerful opponents (based on the actual ranking value of all of each opponents creatures), so alternatively the entire RANK sum for an individual creature could then be:

* (# Affiliated Targets / # Targets) * FOCUS_ON_SINGLE_OPPONENTS *(not yet defined)*

Incorporate "rescue" like behavior for companions who are potentially engaged - with a factoring on ATTENTION_TO_ENGAGED_COMPANIONS. This was lost somewhere along the way - behavior may be emergent from other characteristics.

Note: see "Constants" section for description (and effect) of each constant.

RankTargetsLocally()

This will be a re-sort (by adding a local rank score to the existing rank and re-sorting - i.e. if the creature in position 1's rank score is orders of magnitude in score above position 2+, little will prevent all creatures from attacking (locally ranking at #1) that target.

Targets are re-ranked locally to obtain a target to attack (or to move to attack if out of range).

As per step 8. in the AI Player's move, this is done for every creature (AI Creature) for each target creature (Target Creature) in the RankedTargets list.

```
Local Rank for creature x =
      (Rank
       + IsEngaged * (Move Proximity == 0) * FOCUS_ON_ENGAGING_TARGETS
       + Threat(Target Creature, AI Creature) * CREATURE_SELFISHNESS
       + Threat(AI Creature, Target Creature) * DESIRE_TO_KILL
       + (Target == Last Target) * CREATURE_DETERMINATION
       + (1 / Team Compatibility) * ATTENTION_TO_DIFFICULT_TARGETS
      ) * Compatibility
```

The Compatibility multiplier essentially forces unattackable targets to the bottom of the list when Compatibility is 0 and as a consequence of wizards being prone to attacks from all creatures (and therefore higher in the list) mean a creature should never attack an incompatible target.

Must include significant weighting for any immediately threatening creatures (i.e. ignoring tactics in favor of self defense) - NOTE: this replaces the old AssessThreatsToCreature() method and associated break-out loop from inner list of step creatures.

SelectAndCastSpell

Expand, considering:
- value ranking (against ranked targets)
- relative value of subsequent creature
- tactical (group wise) value
- likelihood of success (factored against constant)
- forward value (i.e. spell tactics)
- likelihood of illusion present (to cast disbelieve) (simple decision tree?)
- decision on casting an illusion (simple decision tree?)
- casting creatures that will be incompatible (undead etc)
- decision on non-creature spells and buffs, inc blobs, fires, citadels, turmoil, subversion, vortex, shadow form, teleport (embedded within decision tree that includes creature spells?)

TestEngagement()

Test for engagement and store in IsEngaged

MeleeAttack()

Attack the CurrentTarget with this creature's melee attack.

If out of range, this method could still be called (but obviously no attack will take place).

RangedAttack()

Attack the CurrentTarget with this creature's ranged attack.

If out of range, this method could still be called (but obviously no attack will take place).

MoveToTarget()

Make a movement decision for the current creature, based on it targeting the specified target.

If engaged, MoveToTarget() will return without movement having happened.

Movement assumes doing a pathfind to desired position and moving through that route as many place as is permitted by creature's movement ability.

Does not consider terrain types (other than in the sense that they're passable by some creatures and not others) - i.e. a defensive move for a weak creature vs one that can't pass terrain x, would be to move into terrain x and attack ranged only.

Also does not consider avoiding engagement and tactical choices on which side to engage an enemy, blocking opponent paths etc.

OPTION 1

If the creature has a melee attack, move to within Melee Range (which may mean staying put).

If the creature only has a ranged attack, either:

Stay put if in range

Move to within range (but no further)

OPTION 2

Make assessment of relative threat and either act:

AGGRESSIVELY

If creature has melee move to within melee

If no melee, move toward target (expand - how far toward target?)

DEFENSIVELY

Regardless of melee ability, move to just within ranged range

SCARED

Move away from target (which may still be in ranged range)

PurgeTargets

Check each creature / wizard in RankedTargets list still exists (if not, remove it from list)

Threat (attacker, defender)

The relative threat imposed by creature a on creature b.

EXPAND CALCULATION

Must sum (or multiply as appropriate):

- Actual Attack (a, d)

- Actual Defense (a, d)
- Proximity for Melee (in terms of number of turns to reach range - 0 should be special case)
- Proximity for Ranged (in terms of number of turns to reach range - 0 should be special case)
- Chance of hit (is this applicable)
- Chance of resist
- Chance of engagement and/or being engaged

Actual Attack (attacker, defender)

The actual attack value of creature a when attacking creature b.

EXPAND

Includes:
- base attack value
- buffs
- terrain bonus for attacker (inc special terrain types - citadel etc - if applicable)

Actual Defense (attacker, defender)

The actual attack value of creature a when attacking creature b.

EXPAND

Includes:
- base defence value
- buffs
- terrain bonus for attacker (inc special terrain types - citadel etc - if applicable)

Note: most are methods that return a scalar.

Relative Value

Typically used for AI player's creatures. The value of a creature compared to other creatures.

Calculated as a sum of relevant creature attributes with an individual multiplier constant on each. Attribute sum may look like:

```
RELATIVE_VALUE=
        movePoints * IMPORTANCE_OF_AGILITY
      + attack * IMPORTANCE_OF_ATTACK
      + defence * IMPORTANCE_OF_DEFENCE
      + magicdefence * IMPORTANCE_OF_DEFENCE
      + initiative * IMPORTANCE_OF_AGILITY
      + canFly * IMPORTANCE_OF_AGILITY
      + isUndead * IMPORTANCE_OF_UNDEAD
      + isMount * IMPORTANCE_OF_AGILITY
```

Note: the IMPORTANCE_OF_UNDEAD is no more complicated than the AI Player favoring undead creatures in spell selection and tactics - however, see alternative note 2 below where it could be more significant.

ALTERNATIVE 1: a finer grained weighting could apply constants to each part of the calculation (i.e. split out agility and the like).

ALTERNATIVE 2: a more complicated calculation could be based upon game state, e.g. other creatures on the map (e.g. an undead creature is worth more if there are no other undead creatures, for example).

Note: Could be calculated differently per AI Type, but shouldn't need to be directly - changes in constants should be sufficient.

Perceived Value

The apparent value of an opponent creature or wizard. Similar to Relative Value, but:

- Ignorant of illusions
- Doesn't know stats on wizard
- Wizards' value will be multiplied by a fixed (per AI Type) constant PERCEIVED_WIZARD_IMPORTANCE such that they are more directly targeted.

Note: A separate set of IMPORTANCE_OF_ constants could be used for perceived value (for each AI Type) but is unlikely to be needed.

Compatibility

The ability for the given creature to attack (in any form) the given target. 0 if incompatible (i.e. attacking undead with a non-magic-casting mortal), 1 otherwise.

If water creatures are invunerable to attack (other than magic, other water creatures and dive bombs) this would include return 0 in such a circumstance.

The logic for any such compatibility test (or incompatibility between creature types / situations) can be embedded here without modifications elsewhere.

Wizards are always 1 against any target.

Team Combatibility

The team's mean compatibility with the target (3 of 4, including wizard would be 0.75).

Attention from Opponent

Relatively, how attention is the AI Player getting from the give opponent.

Could be calculated (remembered) as:

> Total Number of Opponent Attacks / Total Number of Opponent Attacks Against Me

Note: A human player will likely make a judgement decision on this, not an actual one - but remembering the attack distribution for the purposes of satisfying the above sum for AI Players shouldn't have a significant unfair impact.

AI Type

A set of fixed weightings and constants that characterise a particular AI. See

"Weighting Constants" in the Constants section.

Simple Distance

Straight-line distance between creatures.

ALTERNATIVE: use Move Proximity (far more potential computation due to aggregate use of Simple Distance in IdentifyTargets).

Melee Range

Generally 1, but greater in some cases (creatures with Dive Bomb ability, for example).

Move Proximity

Number of moves (spaces) required to be adjacent to target. If the target's already adjacent, should return 0.

If it's not possible to reach the target (i.e. blocked or incompatible terrain), return PROXIMITY_INFINITE.

Number of moves based on path find to target, not straight line.

Proximity should include adverse terrain affects (i.e. a single hex width 50% movement speed will result in a distance of 2, not 1, for that hex). Note - irrelevant if moves are costed.

Melee Attack Proximity

=Move Proximity - Melee Range (floored to zero - negative results only indicate creature is already in range).

Note: not accurate/optimal under certain conditions (i.e. dive bombing could bomb over impassable terrain, meaning a move pathfind takes a longer route).

Ranged Attack Proximity

Based on getting to a point from which a ranged attack can be initiated.

For simple calculation, it could be:

= Move Proximity - Ranged Range

However, this doesn't provide an optimal path as blocking in the later section is different for ranged weapons and creature movement.

Melee Defense Proximity

=Move Proximity - Melee Range of targeted creature (floored to zero - negative results only indicate already in range of that creature).

Note: not accurate/optimal under certain conditions (i.e. dive bombing could bomb

over impassable terrain, meaning a move pathfind takes a longer route).

See note 1. regarding AI Player's knowledge of other creatures' abilities.

Ranged Defense Proximity

Based on the targeted creature getting with ranged distance.

For simple calculation, it could be:

= Move Proximity - Ranged Range of targeted creature

However, this doesn't provide an optimal path as blocking in the later section is different for ranged weapons and creature movement.

See note 1. regarding AI Player's knowledge of other creatures' abilities.

Turn Age

The number of turn for which a creature has been present in the game.

Boolean values stored as int(0|1) for the purposes of calculation.

RankedPOPs - see RankPOPs - top of list is greatest value (Relative Value - with Wizard at top regardless)

Targets - list of targets (see IdentifyTargets)

RankedTargets - ranked list of targets (see RankTargets)

CurrentTarget - (per creature) the target a creature is currently operating on

RankedLocalTargets - re-ranked version of RankedTargets based on applicability to current creature

IsEngaged (bool) - is current creature currently engaged

Constants

See AI Type for details on usage

Note: each constant may require a companion constant (multiplier) (not specific to AI Type) in order to balance disparate figures and obtain approximate balance amongst design-adjusted constants (those below) such that they can all be represented in similar orders of magnitude and/or are otherwise clear in usage. A range of viable/sensible parameters should come out in testing (and analysis of ranking functions outputs).

General constants

MAX_TARGETS - The max number of targets considered in AI players decision

PROXIMITY_INFINITE - Likely 999 or similar.

Weighting constants (all are relative - i.e. increasing every constant by the same factor will have no affect).

PERCEIVED_WIZARD_IMPORTANCE - Higher values will cause AI Players to target player wizards more directly

ENGAGEMENT_PARANOIA - Higher value will cause creatures that can't be engaged (no adjacent enemies) to act first.

FOCUS_ON_ENGAGING_TARGETS - Higher values will cause creatures to directly target any adjacent creatures when engaged, lower values won't necessarily mean the oppposite but will allow for a chance of attacking other creature with ranged attacks if other circumstances promote/permit it.

ATTENTION_TO_ENGAGED_COMPANIONS - Higher value will cause more attacks on creatures that are potentially engaging companions.

IMPORTANCE_OF_AGILITY - Used in relative value. Will cause AI Player to favor agile creatures in spell selection and tactical importance.

IMPORTANCE_OF_ATTACK - Used in relative value. Will cause AI Player to favor high attack value creatures in spell selection and tactical importance.

IMPORTANCE_OF_DEFENCE - Used in relative value. Will cause AI Player to favor high defence value creatures in spell selection and tactical importance.

IMPORTANCE_OF_UNDEAD - Used in relative value. Will cause AI Player to favor undead creatures in spell selection and tactical importance.

TARGET_NEARBY_CREATURES - Higher value mean creatures that are closer to more of the AI Player's creatures will be targeted more readily.

WIZARD_PROTECTION - Higher values will cause all creatures to target creatures that threaten the wizard most.

COMPANION_PROTECTION - Higher values will distort targets such that the most threatening creatures (to the entire party) are ranked at the top.

PARTY_DETERMINATION - The extent to which targeting choices should be carried between turns. High values will cause creatures that are ranked high to continue to rank high until they are killed and new creatures to be ignored.

ATTENTION_TO_NEW_CREATURES - Higher values will cause newly casted creatures to be ranked higher. Can be used to balance PARTY_DETERMINATION

FOCUS_ON_AGGRESORS - Higher values will cause higher ranking of creatures belonging to opponents who have attacked this AI Player more than other players, and subsequently should cause opponents who have not attacked this AI Player to be left alone (all else being equal).

CREATURE_SELFISHNESS - The attention paid to creatures who impose a local

(personal) threat to the creature in question. Essentially the extent to which tactics are followed.

DESIRE_TO_KILL - Higher values will cause higher targeting of creatures that this AI Creature has a better chance of killing. THIS IS IMPORTANT as it effectively dicatates the sensible matchmaking of creatures in the AI Player's roster with the tactical list of targets). Obviously needs to be balanced against everything else though ;)

CREATURE_DETERMINATION - Higher values will cause the creature to prefer creatures that it's previously targeted.

ATTENTION_TO_DIFFICULT_TARGETS - Higher values will cause creatures that are compatible with creatures that are difficult (in the sense than not many companions can target then - i.e. undead) to target them more directly. I.e. undeads will always go for undeads all else being equal.

Gareth Jenkins | Post a Comment | Share Article | Permalink | tagged ai, chaos, creativecommons, design, resource in resources

[illegible]

Monday, September 13, 2010 at 10:43PM

This is the third in a series of free sprite sheets we're putting up here for any use, under a Creative Commons attribution license. It continues the warrior / fighter theme - this time with an eastern twist. For previous sprite sheets see:

- 35 free fantasy creature designs and sprite sheets
- Free modern soldier / war fighter sprite sheet

So, below you'll see links for the sprite sheets (in large 256px and small 128px versions), as well as the original layered PSD art for a Ninja / Shinobi spritesheet. A Ninja was a Japanese mercenary who specialized in unorthodox forms of war. Unlike the Samurai, the Ninja operated covertly.

The Ninja was trained in espionage, sabotage, assassination and various countermeasures. The sprite sheet we're giving away here includes animations for Tai sabaki (silent walking), an armed attack / strike with a ninjato (a type of katana) and a shuriken throw.

The various resources for the Ninja sprite sheet are:

- The original layered Ninja Sprite artwork PSD
- The large Ninja sprite sheet (256px) PNG
- Reference XML plist coordinates for the large Ninja sprite sheet (256px)
- The small Ninja sprite sheet (128px) PNG
- Reference XML plist coordinates for the small Ninja sprite sheet (128px)

Feel free to mix up the frame order as you see fit, but the best-fit sequences for walking and striking are as follows:

- walking: w1, w2, w3, w2, w1, stood, w4, w5, w6, w5, w4
- striking: s1>s8, s4, s3, s2, s1, stood

All the files above are free to use for whatever purpose, licensed as per the terms of the Creative Commons attribution license.

For those interested in the technical side of this, check out the modern soldier post for information on how we generated the sprite sheet and index in Zwoptex, as well as pointers to further resources on using these in Cocos2d.

36peas | Post a Comment | Share Article | Permalink | tagged creativecommons, japan, katana, ninja, shinobi, shuriken, sprite in resources